Automated Review Rating System

balanced model + results

Contents

ABSTRACT

The Automated Review Rating System (Balanced Model) automatically predicts customer review ratings (1–5 stars) using Natural Language Processing (NLP) and Machine Learning. To ensure equal representation across all rating categories, a balanced dataset was created by sampling exactly 2000 reviews from each rating class using random sampling with a fixed random state for reproducibility. This resulted in a dataset of 10,000 samples (2000 per class). Each review was cleaned, preprocessed, and converted into numerical features using TF-IDF vectorization. Machine learning algorithms were then trained on this dataset to classify reviews into their respective rating categories. The balanced dataset helps minimize model bias, improves fairness across all classes, and enhances the system's ability to produce reliable and consistent predictions. This version demonstrates the benefits of balanced data in achieving stable model performance.

GitHub Setup

The complete project has been uploaded to GitHub repository for version control and easy access. This repository contains all the files related to the Automated Review Rating System. The repository also contains the documentation explaining each step of the project.

GitHub Repository Link: *Automated Review Rating System*

Environmental Setup

For implementing the Automated Review Rating System project, the environment was set up using Anaconda and Jupyter Notebook. Anaconda was used because it provides an easy way to manage Python libraries and virtual environments. The project was developed using python with the following main libraries installed:

- Python 3.12 (through Anaconda)
- Jupyter Notebook – for writing and running code interactively
- Pandas – for data handling and cleaning
- NumPy – for numerical operations
- Matplotlib and Seaborn – for data visualization
- Scikit-learn (sklearn) – for preprocessing, vectorization (TF-IDF), and train-test splitting
- NLTK – for basic text preprocessing

Data Collection

In this project, data was collected from online source Kaggle. The collected data included customer reviews and their corresponding ratings from different domains like Amazon products, hotels, and restaurants (Zomato). These datasets were chosen to get a wide variety of review texts and sentiments. All the datasets were combined into a single file containing three main columns as; reviews, overall rating, and review length.

Datasets Used:

1. **Amazon Review Dataset**

   Source: amazon-review

   The dataset contains product reviews posted by customers on Amazon. Each review includes a detailed description of the product experience and an overall rating from 1 to 5. Each record in the

dataset includes detailed feedback from customers, aiding in product evaluation and market research.

2. **Book Reviews on Amazon**

Source: trending-books-reviews

This dataset offers an in-depth look into Amazon's top 100 Bestselling books along with their customer reviews, Ratings, Price etc.

3. **TripAdvisor Hotel Reviews**

Source: tripadvisor-hotel-reviews

This is a dataset of hotel reviews from the popular booking website TripAdvisor. This dataset contains 6,444 hotel reviews, each rated on a five-star scale.

4. **Zomato Reviews & Rating Dataset**

Source: zomato-reviews-ratings

This dataset contains restaurant reviews collected from Zomato. Each entry includes a textual review and a corresponding rating (1–5). Helpful in understanding text sentiment in the food service domain.

5. **Amazon Fine Food Reviews**

Source: amazon-fine-food-reviews

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews. Reviews include product and user information, ratings, and a plain text review. Additionally collected to balance the datasets.

These datasets are merged together and formed the final dataset to preprocess and visualize. The dataset contains raw, uncleaned reviews with missing values, duplicate entries, and mixed rating formats. Visualizations are performed on this uncleaned dataset to understand initial patterns and distributions before preprocessing.

The structure of dataset before cleaning:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30934 entries, 0 to 30933
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   review text    30929 non-null  object
 1   overall        30934 non-null  int64 
 2   review_length  30934 non-null  int64 
dtypes: int64(2), object(1)
memory usage: 725.1+ KB
```

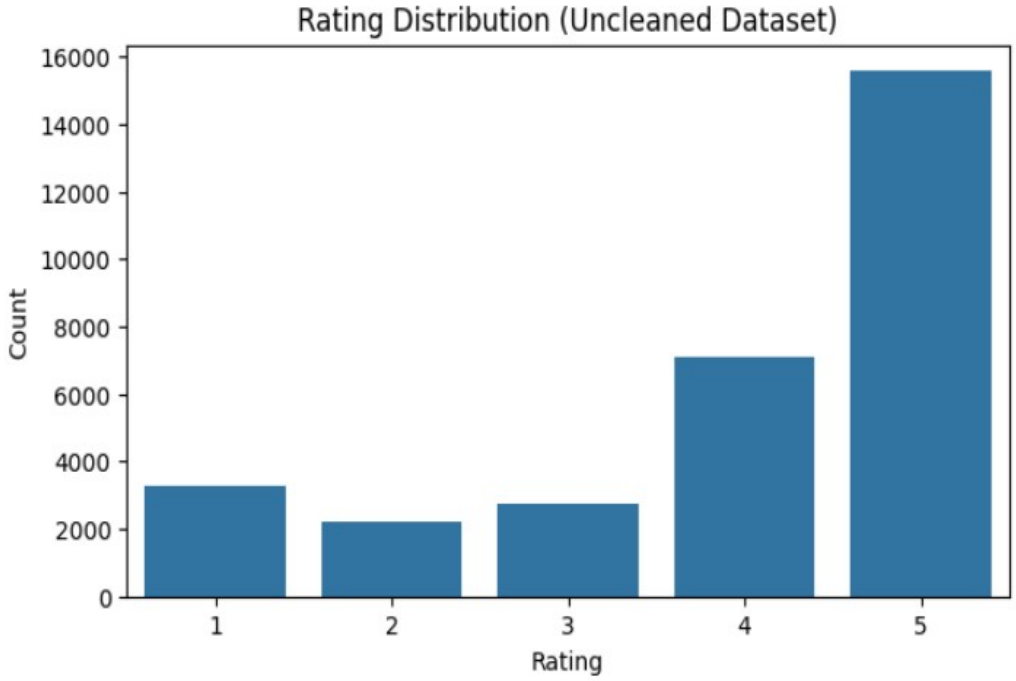
	review text	overall	review_length
0	No issues.	4	2
1	Purchased this for my device, it worked as adv...	5	31
2	it works as expected. I should have sprung for...	4	31
3	This think has worked out great.Had a diff. br...	5	66
4	Bought it with Retail Packaging, arrived legit...	5	52

Data Visualisation of Uncleaned dataset

Representing data graphically to uncover patterns, trends, and insights that are not immediately obvious in raw data.

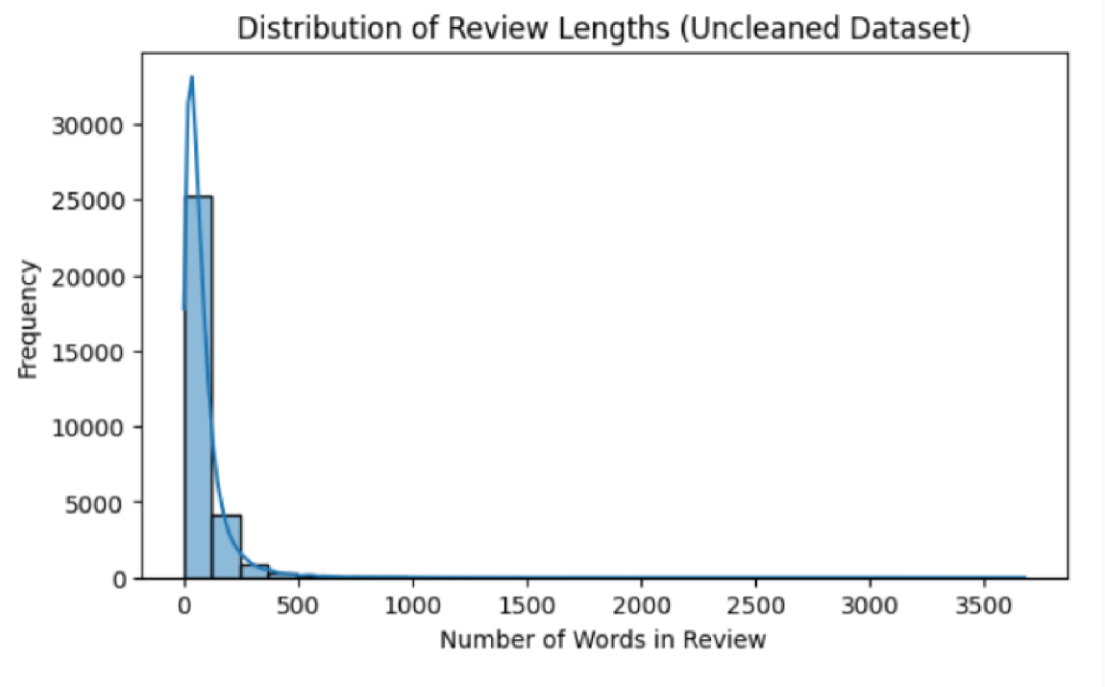
1) *Rating Distribution (barchart):*

```
plt.figure(figsize=(7,4))
sns.countplot(x='overall', data=df)
plt.title('Rating Distribution (Uncleaned Dataset)')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```



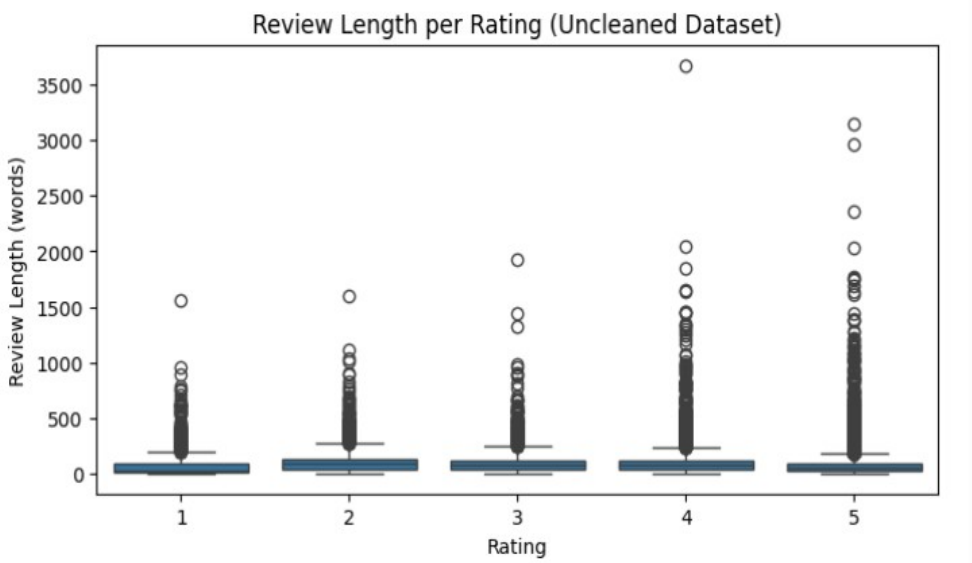
2) *Review length distribution (Histogram):*

```
plt.figure(figsize=(7,4))
sns.histplot(df['review_length'], bins=30, kde=True)
plt.title('Distribution of Review Lengths (Uncleaned Dataset)')
plt.xlabel('Number of Words in Review')
plt.ylabel('Frequency')
plt.show()
```



3) *Boxplot - Review Length per Rating :*

```
plt.figure(figsize=(8,4))
sns.boxplot(x='overall', y='review_length', data=df)
plt.title('Review Length per Rating (Uncleaned Dataset)')
plt.xlabel('Rating')
plt.ylabel('Review Length (words)')
plt.show()
```



Data Cleaning / Preprocessing

After merging all five datasets, the combined dataset underwent several preprocessing steps to ensure the data is clean, consistent, and ready for analysis.

- This involved removing null and duplicate entries, converting text to lowercase, and eliminating unwanted characters such as punctuation, numbers, special symbols, URLs, and extra spaces. After cleaning, the dataset became more consistent and suitable for visualization and further analysis.

1. Missing values:

To identify and remove any missing or incomplete records that could affect the analysis. There's 5 missing values in review column.

```
#missing values
print("\nMissing Values per Column:")
print(df.isnull().sum())
```

output:

```
Missing Values per Column:
review text   5
overall       0
dtype: int64
```

To remove:

```
df = df.dropna (subset=['review text', 'overall'])
```

Output

```
Missing values:
review text      0
overall          0
review_length    0
dtype: int64
```

2 Removing Duplicates:

Duplicate records can significantly affect the quality and accuracy of data analysis. After handling missing values, the dataset was analyzed for duplicate entries. Duplicates were categorized into two types:

- **Exact Duplicates:** Records with identical review text and rating. In this dataset there is no exact duplicates if any found they were removed immediately.
- **Conflicting Duplicates:** Records that share the same review text but have different ratings. These may indicate inconsistency or labelling errors and were identified for further inspection. There is no need to remove them.

```
exact_duplicates = df[df.duplicated(subset=['review text', 'overall'],
keep=False)]
print(f" Exact duplicates found: {len(exact_duplicates)}")
conflicting_duplicates = (df[df.duplicated(subset=['review text'],
keep=False)].sort_values('review text'))
print(f" Conflicting duplicates found: {len(conflicting_duplicates)}")
```

output:

```
Exact duplicates found: 0
Conflicting duplicates found: 679
```

3. Text Cleaning:

To normalize and standardize the review text so that it becomes suitable for NLP-based analysis and model training.

This step removes noise, corrects formatting, and prepares a clean, meaningful representation of each review.

Steps involved:

- Lowercase all text
- Remove URLs, HTML tags, emojis, punctuation, and special characters
- Remove stopwords (using NLTK or SpaCy)
- Apply Lemmatization or Stemming (any one)
- Filter out reviews with fewer than 3 words and excessively long text

Necessary libraries needed:

import re - for Regular expressions (pattern matching, text cleaning)

```
def clean_text(text):
text = text.lower()
text = re.sub(r"http\S+|www\S+", "", text)
text = re.sub(r"<.*?>", "", text)
text = re.sub(r"[^a-z\s]", " ", text)
text = re.sub(r"\d+", " ", text)#numbers
text = text.encode('ascii', 'ignore').decode('utf-8')
text = re.sub(r"\s+", " ", text).strip() #extra spaces
return text
```

applying to review column:

```
df['reviews']=df['review text'].apply(clean_text)
```

Output:

	review text	overall	review_length	reviews
0	No issues.	4	2	no issues
1	Purchased this for my device, it worked as adv...	5	31	purchased this for my device it worked as adve...
2	it works as expected. I should have sprung for...	4	31	it works as expected i should have sprung for ...
3	This think has worked out great.Had a diff. br...	5	66	this think has worked out great had a diff bra...
4	Bought it with Retail Packaging, arrived legit...	5	52	bought it with retail packaging arrived legit ...

After this step, the existing review column is dropped and the review lengths are recalculated based on the cleaned reviews after preprocessing.

```
df = df.drop (columns = ['review text'])
df ['review_length'] = df ['reviews'].str.split().apply(len)
```

Output:

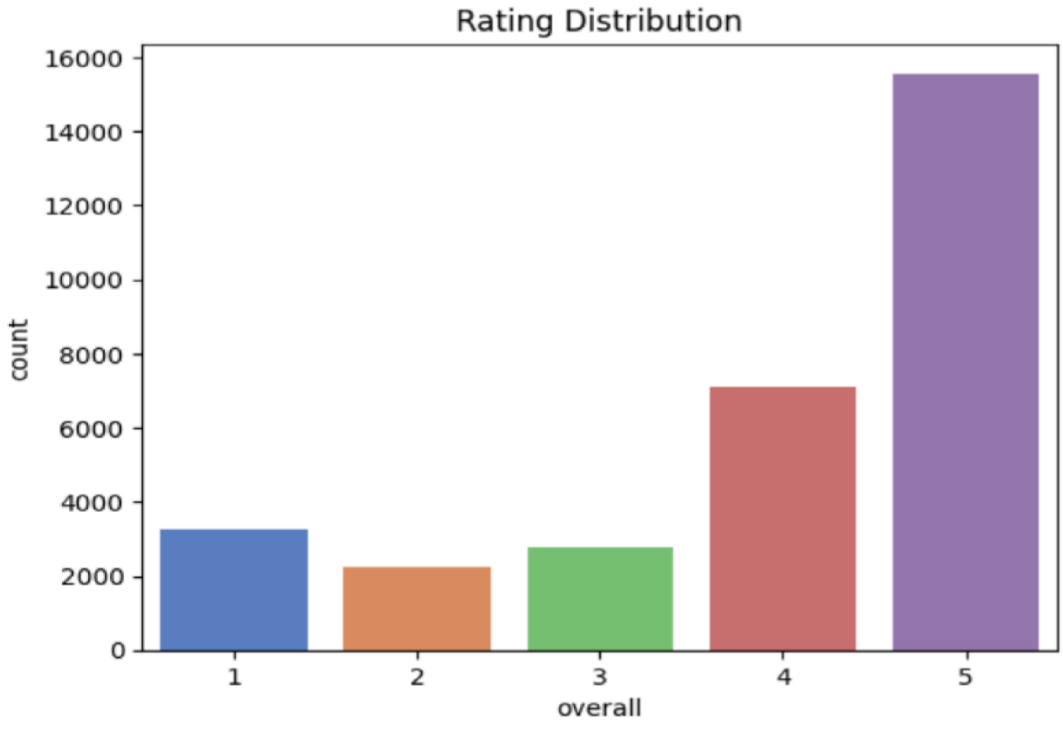
	overall	review_length	reviews
0	4	2	no issues
1	5	31	purchased this for my device it worked as adve...
2	4	31	it works as expected i should have sprung for ...
3	5	70	this think has worked out great had a diff bra...
4	5	51	bought it with retail packaging arrived legit ...

Visualization of cleaned dataset

To analyze how the dataset looks after cleaning and check if the data distribution remains consistent using seaborn.

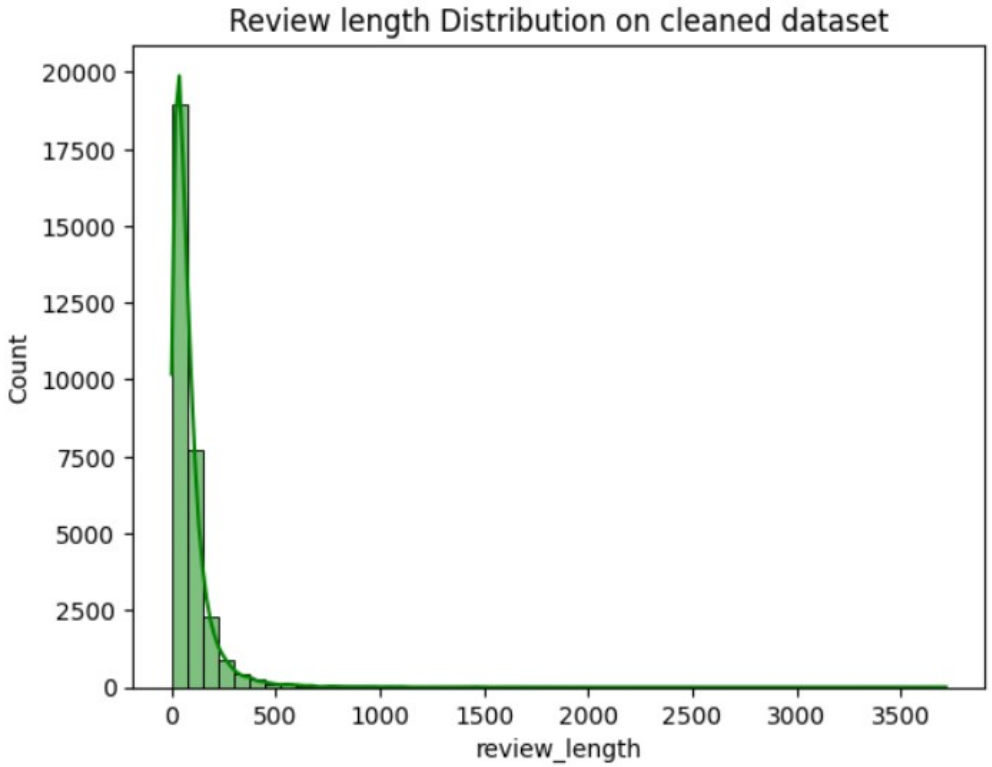
1) Rating Distribution (Bar Chart)

```
sns.countplot(x="overall", data=df, hue="overall", palette="muted",
legend=False)
plt.title("Rating Distribution")
plt.show()
```



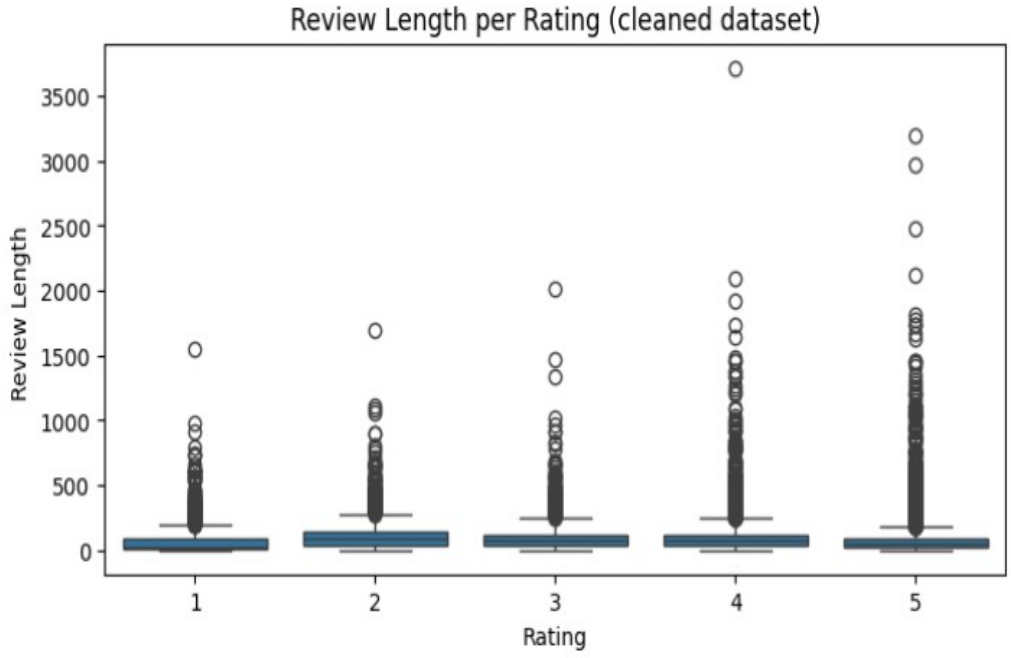
2) *Review length distribution (histogram)*

```
#histogram using seaborn
sns.histplot(df['review_length'], bins=50, kde=True, color="green")
plt.title("Review length Distribution on cleaned dataset")
plt.show()
```



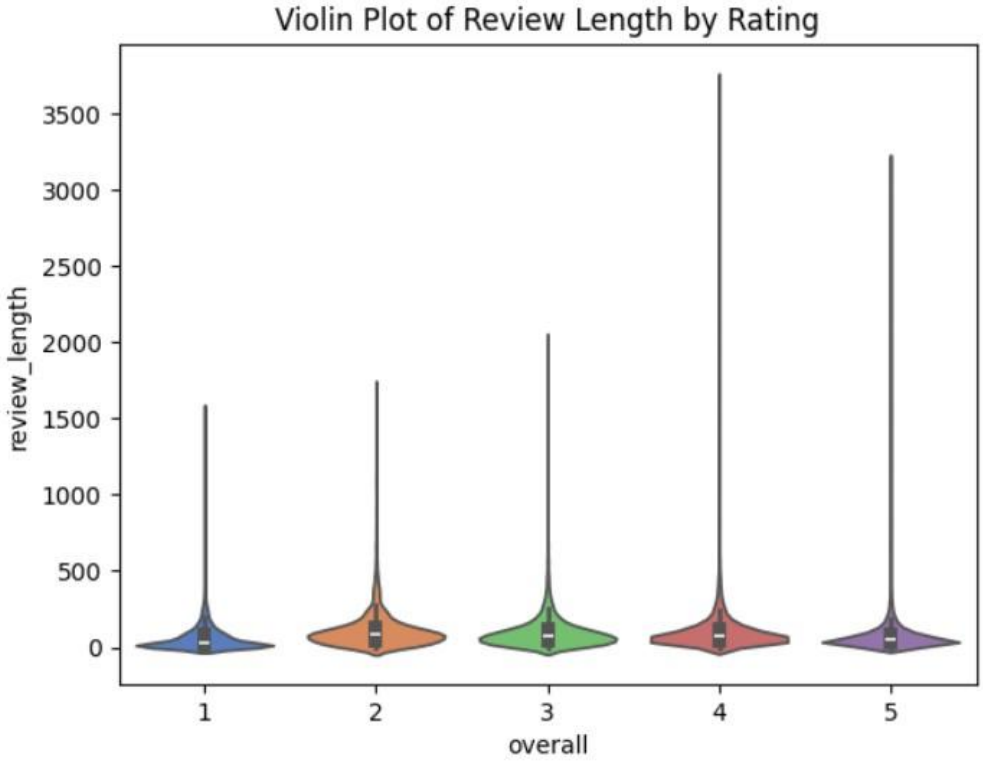
3) *Boxplot of Review Length per Rating:*

```
plt.figure(figsize=(8,4))
sns.boxplot(x='overall', y='review_length', data=df)
plt.title('Review Length per Rating (cleaned dataset)')
plt.xlabel('Rating')
plt.ylabel('Review Length ')
plt.show()
```



4) *Violin Plot (Density + Distribution)*

```
sns.violinplot(x='overall', y='review_length', data=df, hue='overall',
palette="muted", legend=False)
plt.title("Violin Plot of Review Length by Rating")
plt.show()
```



Balanced Dataset Creation

The cleaned dataset initially contained an uneven number of reviews for each rating category. To ensure fair model training, a balanced dataset was created by randomly sampling 2000 reviews from each rating class (1 to 5).

The current rating distribution:

```
print("Original rating distribution:")
print(df['overall'].value_counts())
```

```
Original rating distribution:
overall
5    15569
4     7107
1     3265
3     2763
2     2225
Name: count, dtype: int64
```

Making them balanced with 2000 rows each

```
target_col = 'overall'
balanced_size_per_class = 2000
balanced_samples = []

for cls in sorted(df[target_col].unique()):
   cls_rows = df[df[target_col] == cls]
   sampled = cls_rows.sample(n=balanced_size_per_class, random_state=42)
balanced_samples.append(sampled)

balanced_df = pd.concat(balanced_samples).reset_index(drop=True)
```

The variable target_count = 2000 defines the number of samples to be selected from each rating class.

Output and structure:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   review text    10000 non-null  object
 1   overall        10000 non-null  int64
 2   review_length  10000 non-null  int64
dtypes: int64(2), object(1)
memory usage: 234.5+ KB
```

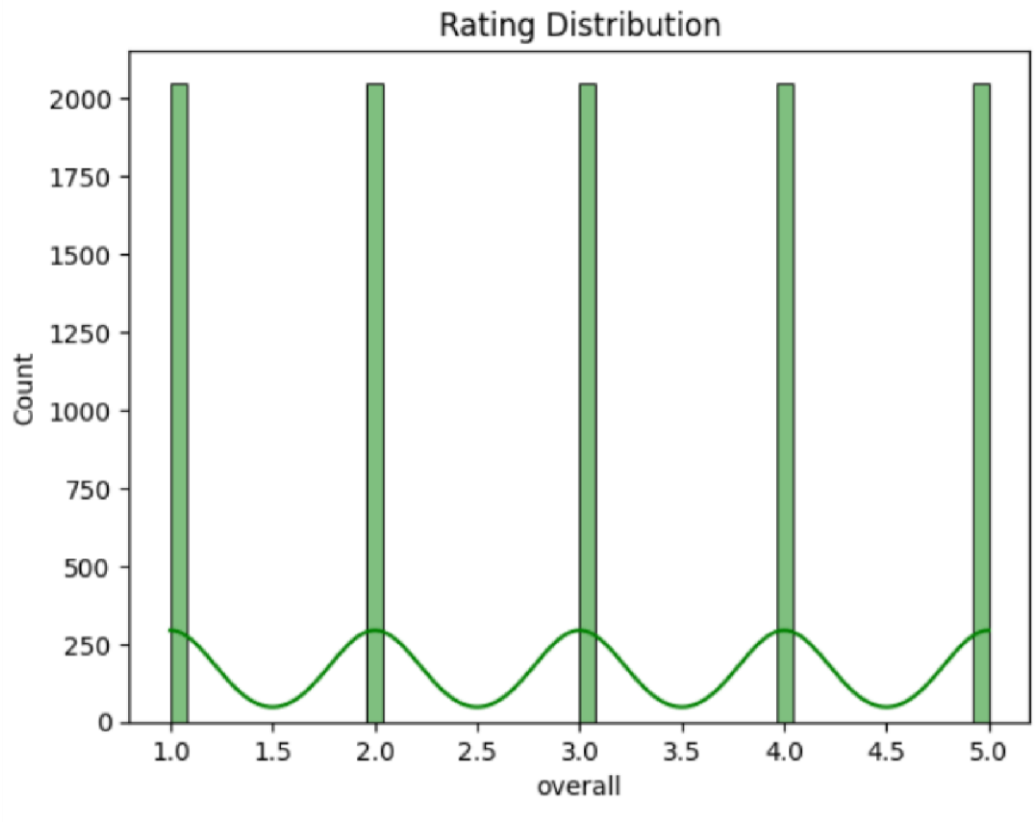
```
print(balanced_df['overall'].value_counts())
```

```
overall
1    2000
2    2000
3    2000
4    2000
5    2000
Name: count, dtype: int64
```

Visualization of Balanced dataset:

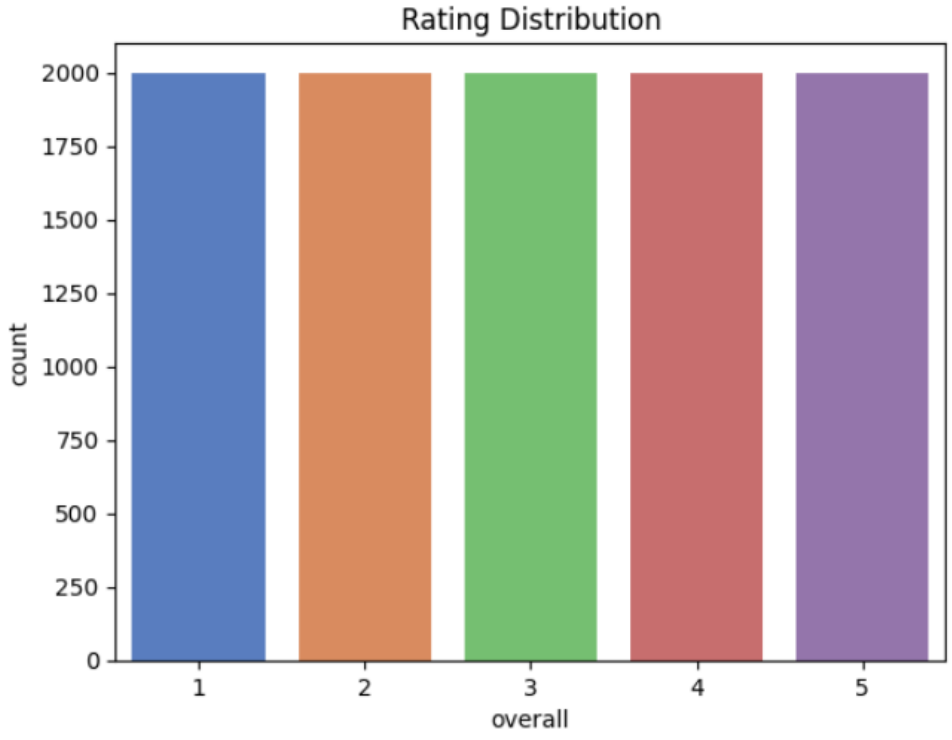
1) Histogram using seaborn (rating distribution):

```
sns.histplot(balanced_df['overall'], bins=50, kde=True, color="green")
plt.title("Rating Distribution")
plt.show()
```



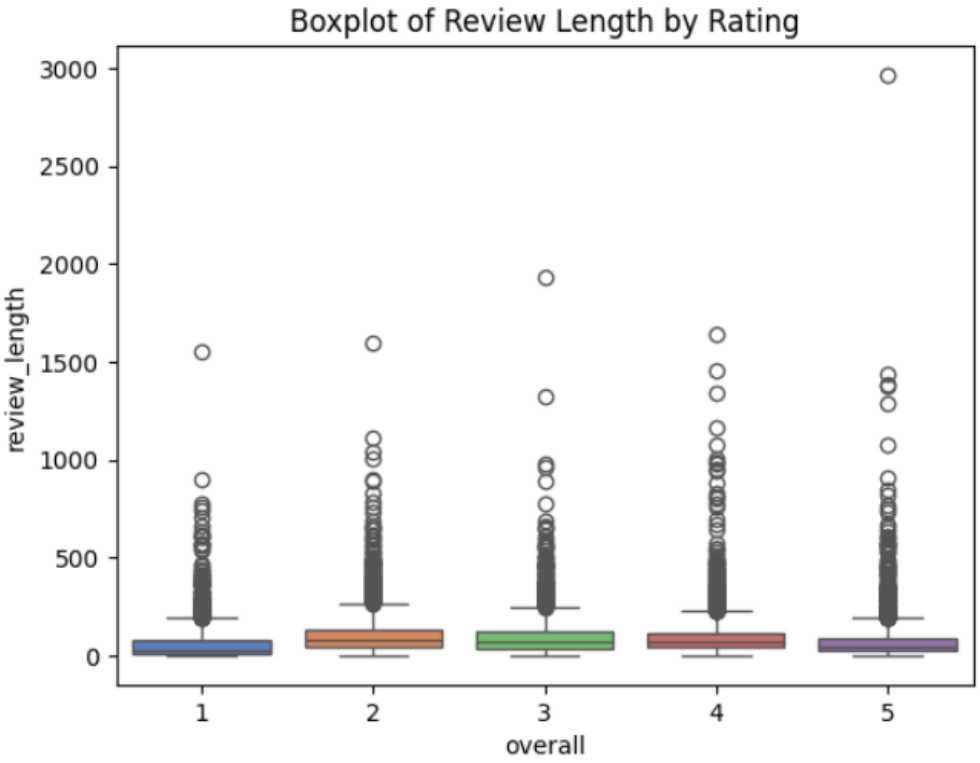
2) *Rating distribution (bar chart):*

```
sns.countplot(x="overall", data=balanced_df, hue="overall", palette="muted", legend=False)
plt.title("Rating Distribution")
plt.show()
```



3) Review Length vs Rating (Boxplot):

```
sns.boxplot(x='overall', y='review_length', data=balanced_df, hue='overall',
palette="muted", legend=False)
plt.title("Boxplot of Review Length by Rating")
plt.show()
```



Tokenization and Stopword Removal:

Stopwords are common words (e.g., *"the"*, *"is"*, *"and"*) that do not carry significant meaning and for this convert each review into individual words (tokens) and remove common, noninformative words(stopwords) to focus on meaningful text for analysis and NLP tasks.

NLTK (Natural Language Toolkit**)** is used for tokenization and stopword removal.

For tokenizing:

```
#spliting string to words
balanced_df['words']=balanced_df['reviews'].apply(lambda x: x.split())
```

This convert the string of words in review column to token of words.

Eg: no issues ⟶ [no, issues]

Output

	overall	review_length	reviews	words
9995	5	96	fantastic treat estherea jewel reception desk ...	[fantastic, treat, estherea, jewel, reception,...
9996	5	39	there are lots of sd memory cards on the marke...	[there, are, lots, of, sd, memory, cards, on, ...
9997	5	21	i purchased this product for my samsung galxy ...	[i, purchased, this, product, for, my, samsung...
9998	5	16	be kind and thoughtful to those left behind lo...	[be, kind, and, thoughtful, to, those, left, b...
9999	5	5	i don t like food	[i, don, t, like, food]

Necessary libraires:

```
Import nltk
from nltk.corpus
import stopwords
```

These libraries from NLTK (Natural Language Toolkit**)** are used for text preprocessing in natural language processing tasks and downloaded stopwords.

```
nltk.download('stopwords')
```

Loading English stopwords and printing them:

```
stop_words = set(stopwords.words('english'))
print("Total Stopwords:", len(stop_words))
print("Stopwords List:", stop_words)
```

```
Total Stopwords: 198
Stopwords List: {'him', 'it', 'this', 'on', 'doing', "they'll", 'who', 'she', 'should', 'yo
u', 'because', 'myself', 'won', "we've", "you'll", "they're", 'through', 'ain', 'more', 'wil
l', 'in', 'where', "you're", 'my', "he'll", 'there', 'y', 'can', "you've", 't', 'theirs', 'he
re', 'had', 'until', 'then', 'was', 'wasn', "it'd", "weren't", 'all', 'ours', 'its', 'her',
"she'd", 'weren', 'at', 'is', 'were', 'when', "he's", "wouldn't", "she's", 'what', 'how', 'm
e', 'they', 'hasn', 'these', 'needn', 'during', 'shan', 'itself', 'aren', "he'd", 'them', 'hi
mself', "i'm", 'haven', 'didn', 'into', 'i', 'ourselves', 'those', "don't", "we'll", 'up', 'b
elow', 've', 'we', 'under', 'again', 'd', 'wouldn', 'has', "won't", 'no', 'out', "you'd", 'wh
y', 'be', 'does', "needn't", 'same', 'your', 'the', 'off', 'yourself', "should've", 'above',
'both', 'to', 'did', 'been', 'while', "mightn't", 'am', "couldn't", 'before', 'about', "i'v
e", 're', 'some', "hasn't", 'few', "didn't", 'themselves', 'so', 'over', 'couldn', 'do', 'mig
htn', 'now', 'too', "doesn't", 'by', 'each', 'whom', 'ma', 'just', 'a', 'and', 'have', "was
n't", 'his', 'any', 'from', 'very', 'than', 'not', 'once', 'being', 'hers', 'only', 'hersel
f', "it'll", 'which', 'an', 'yourselves', 'doesn', 'm', 'further', 's', 'are', "it's", 'tha
t', 'own', 'o', 'having', 'yours', "shan't", 'with', 'or', "i'll", 'our', 'of', 'such', "is
n't", "we're", 'as', 'down', "they'd", "hadn't", 'between', "i'd", "haven't", "she'll", 'he',
'shouldn', "mustn't", 'isn', 'against', 'but', 'mustn', "that'll", "they've", 'nor', "should
n't", 'their', 'other', 'after', "aren't", 'for', 'hadn', "we'd", 'll', 'if', 'most', 'don'}
```

Stopword Removal:

```
def clean_words_list(words):
words = [w for w in words if w not in stop_words]
return " ".join(words)
```

After removing stopwords from the words column, the cleaned tokens were rejoined into strings and saved back into the reviews column for further text processing.

```
balanced_df['reviews'] = balanced_df['words'].apply(clean_words_list)
```

Output:

	overall	review_length	reviews	words
9995	5	96	fantastic treat estherea jewel reception desk ...	[fantastic, treat, estherea, jewel, reception,...
9996	5	39	lots sd memory cards market today cheaper alwa...	[there, are, lots, of, sd, memory, cards, on, ...
9997	5	21	purchased product samsung galxy tab pro far fl...	[i, purchased, this, product, for, my, samsung...
9998	5	16	kind thoughtful left behind loads helpful info...	[be, kind, and, thoughtful, to, those, left, b...
9999	5	5	like food	[i, don, t, like, food]

Why NLTK Was Used?

- NLTK is lightweight and ideal for basic preprocessing tasks like stopword removal, tokenization, and lemmatization.
- SpaCy, while more powerful and faster for large-scale NLP tasks, requires additional model downloads and setup.
- NLTK was preferred for its simplicity, efficiency and ease of integration.

Lemmatization:

Lemmatization is a text preprocessing technique used in Natural Language Processing (NLP) to reduce words to their base or root form, known as a lemma.

example: words like *"running"*, *"ran"*, and *"runs"* are all converted to their base form *"run"*

Necessary libraries:

```
import pandas as pd
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
```

NLTK downloads:

```
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('punkt_tab')
```

- wordnet → provides the base dictionary for lemmatization.
- omw-1.4 → allows WordNet to support multiple languages.
- punkt_tab → would handle tokenization if sentences are separated by tabs.
- punkt → splits text into sentences and words.

Initialising lemmatized and applying lemmatization on tokens:

```
lemmatizer = WordNetLemmatizer()
def clean_words_list(text):
from nltk.tokenize import word_tokenize
words = word_tokenize(str(text))
```

```
    lemmatized = [lemmatizer.lemmatize(w) for w in words]
    return " ".join(lemmatized)
```

Apply back:

```
balanced_df['lemmatized_words'] =
balanced_df['reviews'].apply(clean_words_list)
```

Rechecking length:

```
# Check length of reviews vs lemmatized text
balanced_df['lemmatized_length'] =
balanced_df['lemmatized_words'].apply(lambda x: len(str(x).split()))

balanced_df[['review_length', 'lemmatized_length']].head(10)
```

output:

	review_length	lemmatized_length
0	1	1
1	146	71
2	29	24
3	179	75
4	8	5
5	1	1
6	143	129
7	3	1
8	1	1
9	4	3

	overall	review_length	reviews	words	lemmatized_words	lemmatized_length
9995	5	96	fantastic treat estherea jewel reception desk ...	[fantastic, treat, estherea, jewel, reception,...	fantastic treat estherea jewel reception desk ...	94
9996	5	39	lots sd memory cards market today cheaper alwa...	[there, are, lots, of, sd, memory, cards, on, ...	lot sd memory card market today cheaper always...	26
9997	5	21	purchased product samsung galxy tab pro far fl...	[i, purchased, this, product, for, my, samsung...	purchased product samsung galxy tab pro far fl...	11
9998	5	16	kind thoughtful left behind loads helpful info...	[be, kind, and, thoughtful, to, those, left, b...	kind thoughtful left behind load helpful infor...	9
9999	5	5	like food	[i, don, t, like, food]	like food	2

Dropping unnecessary cloumns:

```
balanced_df = balanced_df.drop(columns=['review_length'])
balanced_df = balanced_df.drop(columns=['words'])
balanced_df = balanced_df.drop(columns=['reviews'])
```

Output:

	overall	lemmatized_words	lemmatized_length
0	1	excellent	1
1	1	taking tour hangzhou china shooting canon card...	71
2	1	sari sabji basi thi rice half bowl bhi nhi tha...	24
3	1	figured yet dummy defective card know card bad...	75
4	1	good delevery thank much partner	5

	overall	lemmatized_words	lemmatized_length
9995	5	fantastic treat estherea jewel reception desk ...	94
9996	5	lot sd memory card market today cheaper always...	26
9997	5	purchased product samsung galxy tab pro far fl...	11
9998	5	kind thoughtful left behind load helpful infor...	9
9999	5	like food	2

Filteration of Reviews

Remove reviews that are too short or too long to ensure data quality.

Printing reviews that are short or equivalent to 3 words and longer than or equivalent to 200 words; (first 10 samples)

```
# Short reviews (length 3-50)

short_reviews = balanced_df[(balanced_df['lemmatized_length'] >= 0) &
(balanced_df['lemmatized_length'] <= 3)]
print("Short reviews:")
print(short_reviews['lemmatized_words'].head(10))

# Long reviews (length 151-200)

long_reviews = balanced_df[(balanced_df['lemmatized_length'] > 150) &
(balanced_df['lemmatized_length'] <= 200)]
print("\nLong reviews:")
```

```
print(long_reviews['lemmatized_words'].head(10))
```

Output:

```
Short reviews:
0             excellent
5                  good
7                  like
8                 thanks
9      smelly pav bhaji
12                 good
14           nice taste
23    delivery guy good
32        packing worst
33     kulcha cook well
Name: lemmatized_words, dtype: object

Long reviews:
18     worst hotel experience thinking booking room h...
105    excellent location cheap bed service sydney mo...
108    horrible experience wished went website booked...
122    disappointed extremely disappointed stay jayak...
153    stay thief stayed boyfriend night spending day...
163    n fooled price location grade room damrak hote...
178    waste vacation read really half star resort ra...
200    vacation disaster breeze punta cana boyfriend ...
221    miss stayed hotel friend weekend th st septemb...
277    awful traveling boyfriend thought dorm amsterd...
Name: lemmatized_words, dtype: object
```

Removing them:

```
balanced_df =
balanced_df[(balanced_df['lemmatized_length']>=3)&(balanced_df['lemmatized_length']<=200)
```

After removing the short and long reviews it found to become imbalanced, the dataset size reduced from 10000 to 8606 reviews as follows;

```
overall
5    1819
4    1783
3    1734
2    1683
1    1587
Name: count, dtype: int64


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8606 entries, 0 to 8605
Data columns (total 3 columns):
```

```
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   overall            8606 non-null   int64
 1   lemmatized_words   8606 non-null   object
 2   lemmatized_length  8606 non-null   int64
dtypes: int64(2), object(1)
memory usage: 201.8+ KB
```

To maintain balance, a new dataset was obtained, cleaned using the same steps (stopwords removal, lemmatization, duplicates and nulls removal, text cleaning), and downsampled to 2,000 reviews per rating (1–5).

This new cleaned data was merged to existing balanced dataset.

Structure of new cleaned data as follows:

	overall	cleaned_reviews	review_length
9995	5	using nupro year pleased result nupro balanced...	34
9996	5	good quality dog eat many dry dog food turn nose	10
9997	5	fed trio min american eskimo french bulldog bl...	43
9998	5	complaint purchase delicious nut shell half wa...	24
9999	5	store manager always interested customer would...	71

```
overall
1    2000
0    2000
1    2000
2    2000
3    2000
Name: count, dtype: int64
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   overall          10000 non-null  int64
 1   cleaned_reviews  10000 non-null  object
 2   review_length    10000 non-null  int64
dtypes: int64(2), object(1)
memory usage: 234.5+ KB
```

The cleaned and balanced new dataset was then combined with the old existing cleaned dataset, aligning the columns (overall, review text, review_length) to create a consolidated dataset ready for analysis and modeling.

Before combining renamed the column names:

```
df_new= df_new.rename(columns={"cleaned_reviews": "reviews"})
balanced_df= balanced_df.rename(columns={"lemmatized_words": "reviews"})
balanced_df= balanced_df.rename(columns={"lemmatized_length": "review_length"})
```

Combining:

```
df_1= df_new[["overall", "reviews", "review_length"]]
df2= balanced_df[["overall", "reviews", "review_length"]]
final_df = pd.concat([df_1, df2], ignore_index=True)
```

output:

	overall	reviews	review_length
18601	5	good hotel stayed hotel recently time june saw...	51
18602	5	fantastic treat estherea jewel reception desk ...	94
18603	5	lot sd memory card market today cheaper always...	26
18604	5	purchased product samsung galxy tab pro far fl...	11
18605	5	kind thoughtful left behind load helpful infor...	9

Structure of final dataset

```
print(final_df['overall'].value_counts())
```

```
overall
5    3819
4    3783
3    3734
2    3683
1    3587
Name: count, dtype: int64
```

Balancing (downsampling):

Downsampling the final dataset to the least rating count that is 3585.

```
min_count = final_df["overall"].value_counts().min()
df_balanced = (
  final_df.groupby("overall", group_keys=False)
        .apply(lambda x: x.sample(n=min_count, random_state=42))
        .reset_index(drop=True))
```

Output and structure of final data:

```
overall
1    3585
2    3585
3    3585
4    3585
5    3585
Name: count, dtype: int64


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17925 entries, 0 to 17924
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   overall        17925 non-null  int64
 1   reviews        17925 non-null  object
 2   review_length  17925 non-null  int64
dtypes: int64(2), object(1)
memory usage: 420.2+ KB
```

Visualization of Final Balanced Dataset

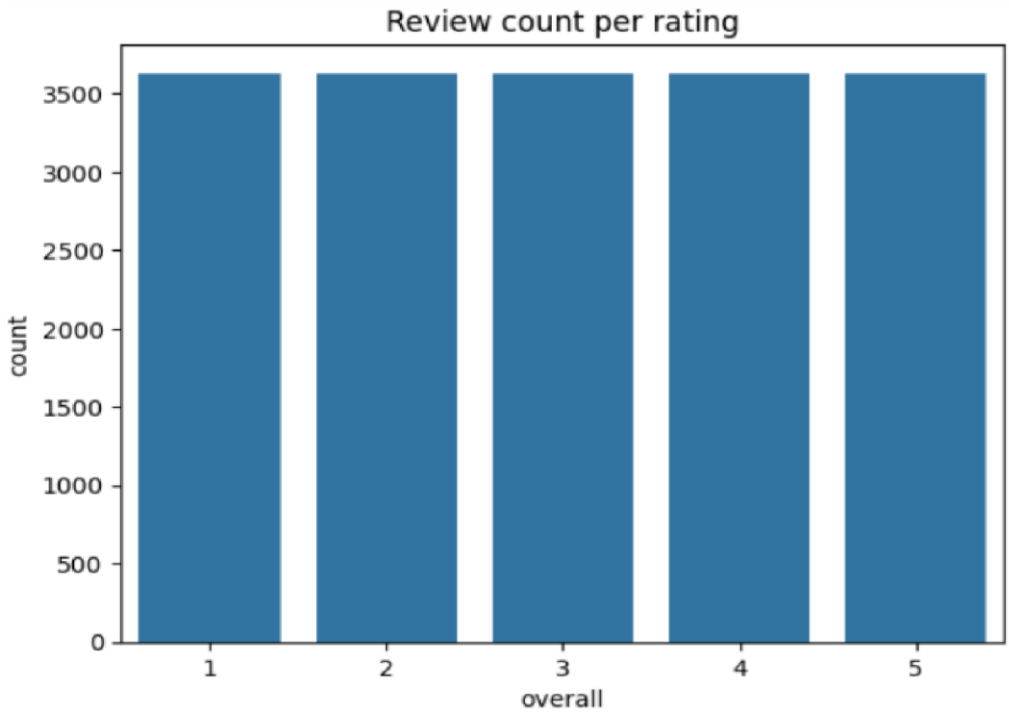
To explore the distribution of reviews and ratings in the consolidated, cleaned, and balanced dataset.

Necessary libraries:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

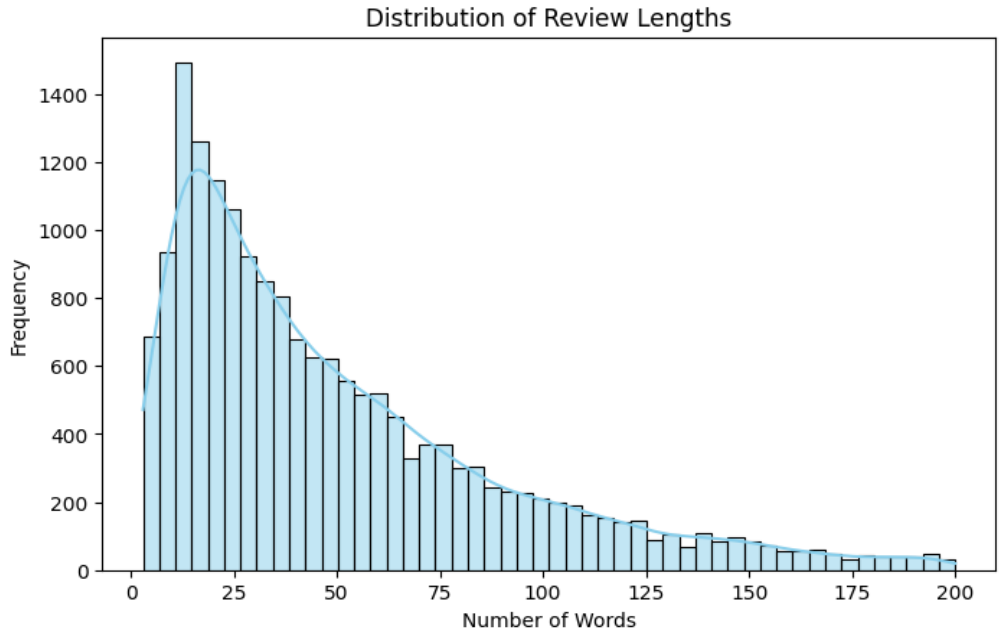
1. *Rating Distribution*

```
sns.countplot(x="overall", data=df_balanced)
plt.title("Review count per rating")
plt.show()
```



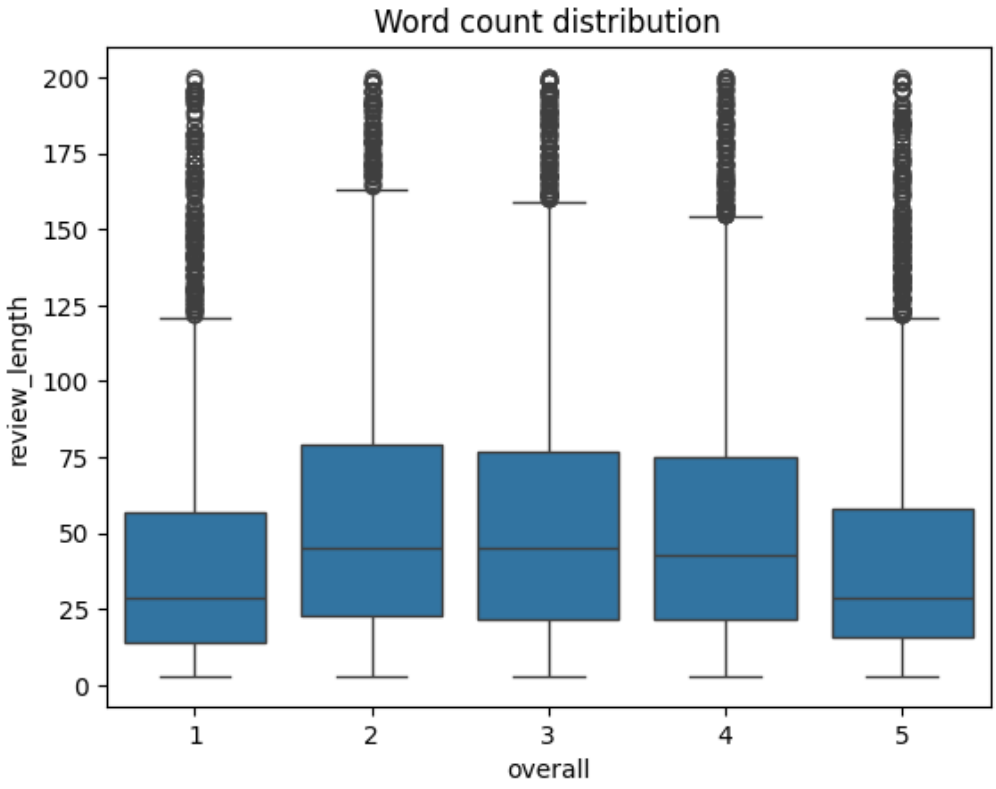
2. *Review Length Distribution*

```
plt.figure(figsize=(8,5))
sns.histplot(df_balanced['review_length'], bins=50,
kde=True, color='skyblue') plt.title('Distribution of
Review Lengths') plt.xlabel('Number of Words')
plt.ylabel('Frequency') plt.show()
```



3. *Boxplot: Review Length vs Rating*

```
sns.boxplot(x="overall", y="review_length", data=df_balanced)
plt.title("Word count distribution")
plt.show()
```



4. *displaying samples:*

```
for rating in sorted(df_balanced["overall"].unique()):
    print(f"\n⋆Rating {rating} Sample Reviews:")
    print(df_balanced[df_balanced["overall"] ==
rating]["reviews"].head(4).to_list())
```

output:

⋆Rating 1 Sample Reviews:

['disappointing booked hotel based review read disappointed booked superior room paying price couple given room big house double bed single travelling companion given single room foot square floor space shower flooded room used complained immediately reception staff unhelpful saying hotel fully booked duty manager able speak day manager changed room descent sized double filthy bath hair nail clipping carpet room bedding arrive handed duvet cover separately left making plus hotel located nice area amsterdam park hotel lovely walk centre hotel close needed refurbishment november got distinct feeling staff lost job properly certainly return', 'horrible ruin name giving type food', 'honey raw represented product label impossible read full web page taste texture definitely heated pale shadow raw star thistle honey', 'ordered pack pay attention opened package realized today received wrong item received langers

cranberry juice cocktail concentrate lot cheaper langers cranberry juice ordered think reason discovered error today ordering langers cranberry juice noticed bottle different going order time pay attention receive next order want say never problem amazon order quite bit varied stuff totally completely fault paying attention received never happen']

⋆Rating 2 Sample Reviews:

['bought putting ice tea dissolve waited waited dissolve next thought would put small bowl put hot tap water melt took around minute dissolve completely long using piping hot coffee would suggest getting stevia either liquid granulated would pas', 'beautiful view disappointing service stayed edgewater th anniversary paid small fortune beautiful waterfront room unfortunately water bay water room apparently replacing plumbing told nothing said reserved checked went upstairs shower change dinner water n flush toilet inquired desk manager said wait awhile sure got water hour got excited got ready shower turned water got brown goop ran awhile cleared hot water inquired told soon couple hour tried lukewarm called desk assured hot finally hotel engineer come room turned water blast stood minute pj agree n hot explanation promise fix thought leaving hotel time late manager nothing offer free breakfast coupon day encountered new manager refunded night nothing really make miserable stay supposed relaxing romantic night away kid stressful miserable stay', 'well seeing received first box olive broken olive juice place give good packing ethic stayed phone hour amazon trying get problem resolved end sent another box time nothing broken see glass bottle wrapped protection thrown box order', 'tried several recipe mix none turn well taste good everything turn crumbles dissapointed']

⋆Rating 3 Sample Reviews:

['slower expected transfer rate used phone tablet expected higher transfer rate given data class', 'nice healthy snack alternative chip cooky even gave daughter since month instead puff moisture change consistency though humid houston need finish pack open', 'expected size small get better bargain store received time love part packaging excellent read carefully size know ordering', 'good booked trip sun wing flight transportation hotel organized time arrived approx people trying check took hour check room nothing special clean lovely view beach block relately quite maid cleaned nicely day mini bar alway supply beer water cola air conditioning safety deposit box worked well sun wing rep amanda inefficient indifferent limited hour resort time area resort clean lot choose buffet food looked inviting actually tasteless coffee good fresh pineapple marvelous certainly personally enjoyed pasta steak house restaurant good italian restaurant passable really n experience communication problem beach beautiful water warm took country adventure tour real eye opener mountain wonderful enjoyed finding growing processing sugar cane vanilla cocao coffee visit school farm house certainly glad canadian tragic people live lunch ranch delicious horse riding pretty rough shape certainly felt guity mounting ride short tour suppose include stop cigar museum cigar happened day trip truck rough came covered grit tour guide alberta entertaining overall enjoyed boyfriend finally time plan return dr decide stay resort plan stay palace']

⋆Rating 4 Sample Reviews:

['like dog food dog high quality expensive say though bag came ragged opened', 'small great booked hotel local travel agent got better rate booking directly hotel note room rate usually increase friday saturday travelled hk ferry macau arrived kowloon ferry terminal amazingly took minute terminal hotel taxi hotel service nice staff speak good english read review room n expect room size actually quite cozy simply fit people necessary equipment packed room however true bed bit small limited space luggage place sleep hour room hotel breakfast fair try difference especially orange juice look breakfast recommend restaurant hotel lot local noodle menu well think good place stay safer staying guesthouse minute walk jordan mtr perfect location', 'using samsung g cell phone memory card expansion work ok allthough card phone slight issue reading massive sd card high compacity hope samsung release software firmware update resolve issue reading card boot phone sometimes found simply restarting phone usually resolve issue reading card error message overall minus slight issue work great', 'gotten germination date happy purchase would order product']

⋆Rating 5 Sample Reviews:
['pretty sure price increased product since last time purchased however exactly want make hummas smooth creamy pleasant taste bitter tahini', 'buy thru amazon tried ebay always come thru got messed four time real recourse lose

paypal nobody pal rack terrible others take advise stick one always came thru every time amazon amazon h rhodes', 'great place away lucky stay drury suite beautifully decorated christmas clean friendly staff great location complaint bathroom guest room small', 'dal makhani mix use result excellent tried different brand indian spice mix available local indian grocer mdh consistently favorite']

Train-Test Split

Dividing the cleaned and balanced dataset into training and testing subsets to train machine learning models and evaluate their performance on unseen data:

- **Training set:** used to train the model
- **Test set:** used to evaluate model performance on unseen data

Necessary libraries

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
```

method:

```
X = df_balanced['reviews']
y = df_balanced['overall']
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

Explanation:

- features (**X**): reviews column (processed/cleaned text)
- labels (**y**): overall rating (1–5)
- split ratio: **80%** training, **20%** testing
- stratification: applied (stratify=y) to preserve class balance in both subsets
- shuffling: enabled (shuffle=True) for randomness
- random seed: random_state=42 for reproducibility

Output:

```
Training samples: 14340
Testing samples: 3585

Class distribution in train set:
overall
2    2868
5    2868
3    2868
4    2868
1    2868
Name: count, dtype: int64

Class distribution in test set:
overall
5    717
2    717
4    717
3    717
1    717
Name: count, dtype: int64
```

Maintains balanced representation of all rating classes.

TF-IDF Vectorization:

Transform cleaned text reviews into numerical feature vectors that represent the importance of each word in a review relative to the entire dataset, making them suitable for machine learning models.

Method:

```
vectorizer = TfidfVectorizer(max_features=10000, stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

print("Train shape:", X_train_vec.shape)
print("Test shape:", X_test_vec.shape)
```

Explanation:

Vectorizer: TfidfVectorizer from sklearn.feature_extraction.text.

Parameters:

- *max_features=10000 → Limits the vocabulary to the 10,000 most frequent words.*
- *stop_words='english' → Excludes common English stopwords. Process:*
- *Training Data (X_train): fit_transform learns the vocabulary and computes the TF-IDF scores.*

- *Testing Data (X_test): transform applies the same vocabulary and weights learned from training data.*

Output:

```
Train shape: (14340, 10000)
Test shape: (3585, 10000)
```

TF-IDF Equation

For a term t in document d in a corpus D:

$$\text{TF-IDF}(t,d,D)=\text{TF}(t,d)\times\text{IDF}(t,D)$$

Term Frequency (TF):

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

Inverse Document Frequency (IDF):

$$\text{IDF}(t,D)= \log \frac{N}{1 + |\{d \in D : t \in d\}|}$$

N → total number of documents in the corpus

$|\{d \in D : t \in d\}|$ → number of documents containing term t

Interpretation:

- High TF-IDF → term is frequent in the document but rare in others (important).
- Low TF-IDF → term is common in all documents or rare overall (less important).

MACHINE LEARNING MODEL TRAINING AND EVALUATION

Evaluates three different machine learning models they are; Logistic Regression, Random Forest, and Support Vector Machine (SVM) on a given dataset for a multi-class classification problem. The goal is to compare the performance of these models using accuracy and classification metrics.

Necessary libraries:

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
```

- *LogisticRegression: A linear model for classification.*
- *RandomForestClassifier: An ensemble model that uses multiple decision trees for improved accuracy.*
- *SVC: Support Vector Machine classifier for finding the optimal hyperplane to separate classes.*
- *accuracy_score: Function to calculate the overall accuracy of predictions.*
- *classification_report: Function to generate precision, recall, F1-score, and support for each class*.

Step 1: Initialize Models

```
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM": SVC(random_state=42)
}
```

Step 2: Train and Evaluate Models

```
results = {}
for name, model in models.items():
    print(f"\n Training {name}...")
    model.fit(X_train_vec, y_train)      # Train the model
    preds = model.predict(X_test_vec)    # Predict on test data

    acc = accuracy_score(y_test, preds)  # Compute accuracy
    print(f" Accuracy for {name}: {acc:.4f}")
    print(classification_report(y_test, preds))

    results[name] = (model, acc)         # Store model and accuracy
```

output:

Model	**Accuracy**
Logistic Regression	0.4703
Random Forest	0.4346
SVM	0.4759

Accuracy indicates that SVM performs slightly better (0.4759) than Logistic Regression (0.4703) and Random Forest (0.4346) on this dataset

Choosed the best model with:

```
best_model_name = max(results, key=lambda k: results[k][1])
best_model = results[best_model_name][0]
print(f"\n Best model: {best_model_name}")
```

best model: svm

Classification report:

```
print(classification_report(y_test, best_model.predict(X_test_vec)))
```

```
              precision    recall  f1-score   support

           1       0.52      0.64      0.57       717
           2       0.39      0.37      0.38       717
           3       0.44      0.38      0.41       717
           4       0.44      0.39      0.41       717
           5       0.56      0.59      0.58       717

    accuracy                           0.48      3585
   macro avg       0.47      0.48      0.47      3585
weighted avg       0.47      0.48      0.47      3585
```

Saving Model and Vectorizer with Joblib:

Code:

```
import joblib

joblib.dump(best_model, "model_B.pkl")
joblib.dump(vectorizer, "vectorizer_B.pkl")
joblib.dump((X_test, y_test), "balanced_test.pkl")

print("saved")
```

Saved the trained machine learning model svm and its associated vectorizer and test data to disk, so they can be reloaded later for prediction without retraining.

CROSSTESTING MODELS

Cross-testing is the evaluation of a model on a dataset different from the one it was trained on.

It includes:

- *Load pre-trained machine learning models (Model_A trained on imbalanced data and Model_B trained on balanced data) and their associated vectorizers.*
- *Load test datasets (imbalanced and balanced).*
- *Transform the test data using the corresponding vectorizers.*
- *Evaluate the models on both own test sets and cross-test sets.*
- *Compare performance using standard metrics: Accuracy, Precision, Recall, F1 (Macro and Weighted).*

1. Libraries Used

```
import joblib
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import pandas as pd
```

2. Loading Models, Vectorizers, and Test Data

```
path = r"C:\Users\hp\Desktop\automatedreviewratingsystem\models"

# Load models and vectorizers
model_A = joblib.load(f"{path}\\model_A.pkl")
vec_A   = joblib.load(f"{path}\\vectorizer_A.pkl")

model_B = joblib.load(f"{path}\\model_B.pkl")
vec_B   = joblib.load(f"{path}\\vectorizer_B.pkl")

# Load test datasets
X_test_A, y_test_A = joblib.load(f"{path}\\imbalanced_test.pkl")
X_test_B, y_test_B = joblib.load(f"{path}\\balanced_test.pkl")

print("models, vectorizers, and test datasets loaded")
```

- *model_A and vec_A → trained on imbalanced data.*
- *model_B and vec_B → trained on balanced data.*
- *Test sets: X_test_A, y_test_A (imbalanced), X_test_B, y_test_B (balanced).*

3. Transforming Test Data Using Vectorizers

```
X_test_A_vec_A = vec_A.transform(X_test_A)  # Model_A on its own test
X_test_B_vec_B = vec_B.transform(X_test_B)  # Model_B on its own test
X_test_B_vec_A = vec_A.transform(X_test_B)  # Model_A on balanced test (cross)
X_test_A_vec_B = vec_B.transform(X_test_A)  # Model_B on imbalanced test (cross)
```

4. Evaluation Function

```
def evaluate(model, X, y):
preds = model.predict(X)
return (
   round(accuracy_score(y, preds) * 100, 2),
   round(precision_score(y, preds, average='macro', zero_division=0) * 100, 2),
   round(recall_score(y, preds, average='macro', zero_division=0) * 100, 2),
   round(f1_score(y, preds, average='macro', zero_division=0) * 100, 2),
   round(f1_score(y, preds, average='weighted', zero_division=0) * 100, 2)
)
```

5. Evaluating Models on Own and Cross Datasets

```
results = []
results.append(["Model_A(imbalanced)", "Test data (own)", *evaluate(model_A, X_test_A_vec_A, y_test_A)])
results.append(["Model_B(balanced)", "Test data (own)", *evaluate(model_B, X_test_B_vec_B, y_test_B)])
results.append(["Model_A(imbalanced)", "Balanced set (cross)", *evaluate(model_A, X_test_B_vec_A, y_test_B)])
results.append(["Model_B(balanced)", "Imbalanced set (cross)", *evaluate(model_B, X_test_A_vec_B, y_test_A)])

summary = pd.DataFrame(results, columns=["Model", "Tested On", "Accuracy %", "Precision(Macro)",
"Recall(Macro)", "F1(Macro)", "F1(weighted)"])


print("\nFinal cross-testing summary table (svm)\n")
display(summary)
```

6. Final Cross-Testing Results

```
Final cross-testing summary table (svm)
```

	Model	Tested On	Accuracy %	Precision(Macro)	Recall(Macro)	F1(Macro)	F1(weighted)
0	Model_A(imbalanced)	Test data (own)	55.91	54.97	52.84	53.65	55.51
1	Model_B(balanced)	Test data (own)	47.59	47.00	47.59	47.08	47.08
2	Model_A(imbalanced)	Balanced set (cross)	45.63	47.19	45.63	44.80	44.80
3	Model_B(balanced)	Imbalanced set (cross)	67.82	70.42	70.80	69.58	67.74

- Model_A performs better on imbalanced data (its own dataset).
- Model_B performs better on imbalanced data when cross-tested (generalization benefit of balanced training).

STREAMLIT APP IMPLEMENTATION

This Streamlit application allows users to input a product or service review and get predicted star ratings from two different machine learning models:

- Balanced Model: Trained on balanced data.
- Imbalanced Model: Trained on imbalanced data.

STEPS AND CODE:

1. Importing Required Libraries

```
import streamlit as st
import joblib
```

- streamlit: Used to build the web UI.
- joblib: Used to load pre-trained models and vectorizers.

2. Loading Models and Vectorizers

```
path = "models"

model_A = joblib.load(f"{path}/fine_tuned_logistic_imbalanced.pkl")
model_B = joblib.load(f"{path}/fine_tuned_svm_balanced.pkl")
vec_A = joblib.load(f"{path}/vectorizer_A.pkl")
vec_B = joblib.load(f"{path}/vectorizer_B.pkl")
```

- The pre-trained models (model_A, model_B) and corresponding text vectorizers (vec_A, vec_B) are loaded from the models directory.

- model_A corresponds to the logistic regression model trained on imbalanced data.
- model_B corresponds to the SVM model trained on balanced data.

3. Styling the Application

```
st.markdown("""
<style>
#MainMenu {visibility: hidden;}
footer {visibility: hidden;}
header {visibility: hidden;}
body{
    background:linear-gradient(135deg,#7b4397,#dc2430);
    color: pink;
    font-family:'Poppins', sans-serif;
 }
.stApp{
    background:#1c1c3c;
    color:#f5f5f5;
    padding:50px;
    border-radius:20px;
    box-shadow:0px 0px 30px rgba(0,0,0,0.5);
    max-width:800px;
    margin:50px auto;
 }
h1{
    color:gold !important;
    text-align:center;
 }
button[kind="primary"]{
    background:gold !important;
    color:black !important;
    font-weight:bold !important;
    border-radius:8px !important;
 }
</style>
 """, unsafe_allow_html=True)
```

Custom CSS to style the app UI:

- Hides Streamlit default menu, footer, and header.
- Sets a gradient background for the page.

- Styles the app container with dark background, padding, rounded corners, and shadow.
- Customizes heading colors and button styles for a look.

4. Setting Page Configuration

```
st.set_page_config(page_title="Automated Review Rating System", page_icon="⭐", layout="centered")
```

- Configures the browser tab title, icon, and centers the layout on the page.

5. Displaying the App Header

```
st.markdown("<h1 style='text-align:center; color:gold;'>⭐ Automated Review Rating System ⭐</h1>", unsafe_allow_html=True)
```

- Shows the app's main title with gold-colored text, centered.

6. User Input

```
user_input = st.text_area("Enter Review Text:", height=150)
```

- Provides a multi-line text area for users to input their review.

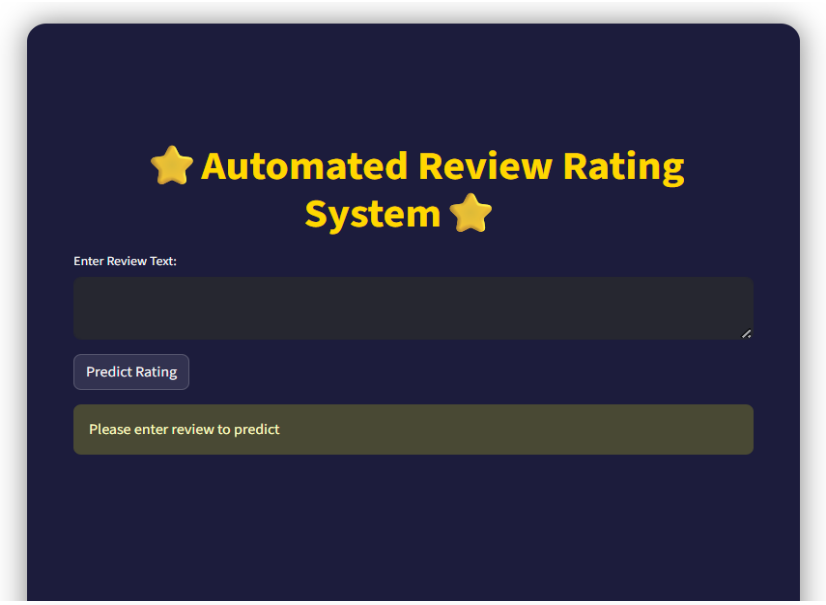
7. Predict Rating Button and Logic

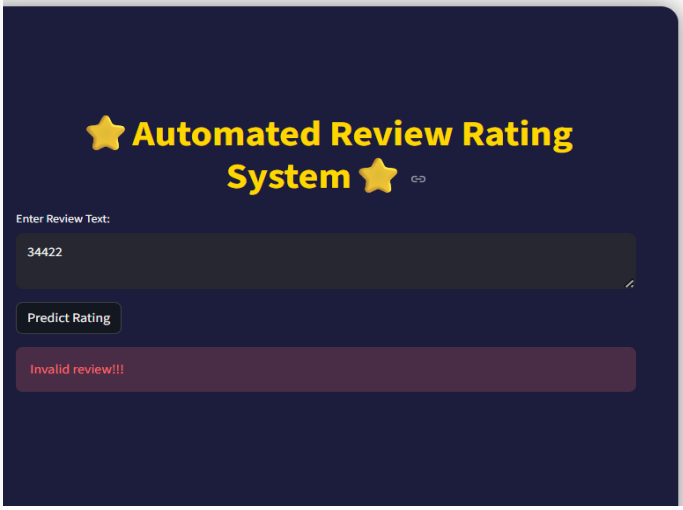
```
if st.button("Predict Rating"):
    review = user_input.strip()
    if review == "":
        st.warning("Please enter review to predict")
    elif not any(char.isalpha() for char in review):
        st.error("Invalid review!!!")
    else:
        X_vec_A = vec_A.transform([review])
        X_vec_B = vec_B.transform([review])

        pred_A = model_A.predict(X_vec_A)[0]
        pred_B = model_B.predict(X_vec_B)[0]

        st.markdown("## Predictions")
        col1, col2 = st.columns(2)
        with col1:
            st.markdown("### *Balanced Model*")
            st.markdown(f"⭐ *{pred_B} Star*")
        with col2:
            st.markdown("### *Imbalanced Model*")
            st.markdown(f"⭐ *{pred_A} Star*")
```

When the user clicks Predict Rating:

- Input is stripped of extra spaces.
- Checks if input is empty; if yes, shows a warning.
- Checks if input contains any alphabetic characters; if not, shows an error.
- Otherwise, transforms the review using the vectorizers.
- Gets predictions from both models.
- Displays predictions side-by-side in two columns:
- Left column: Prediction from the balanced model.
- Right column: Prediction from the imbalanced model.

Output:

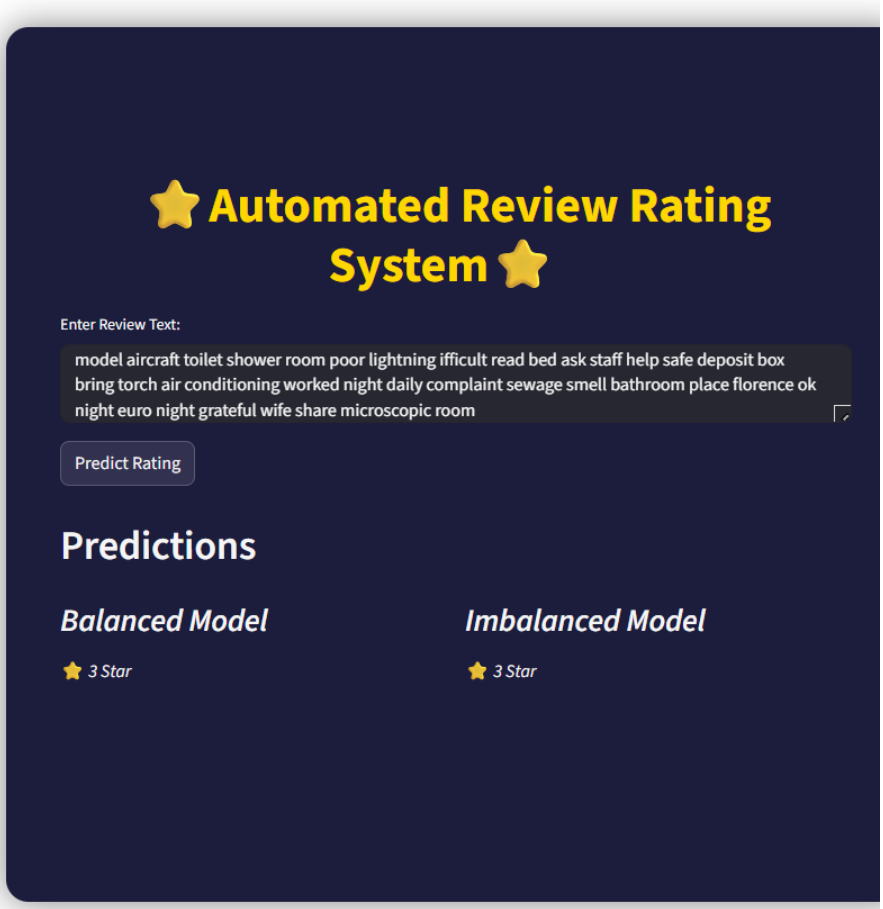


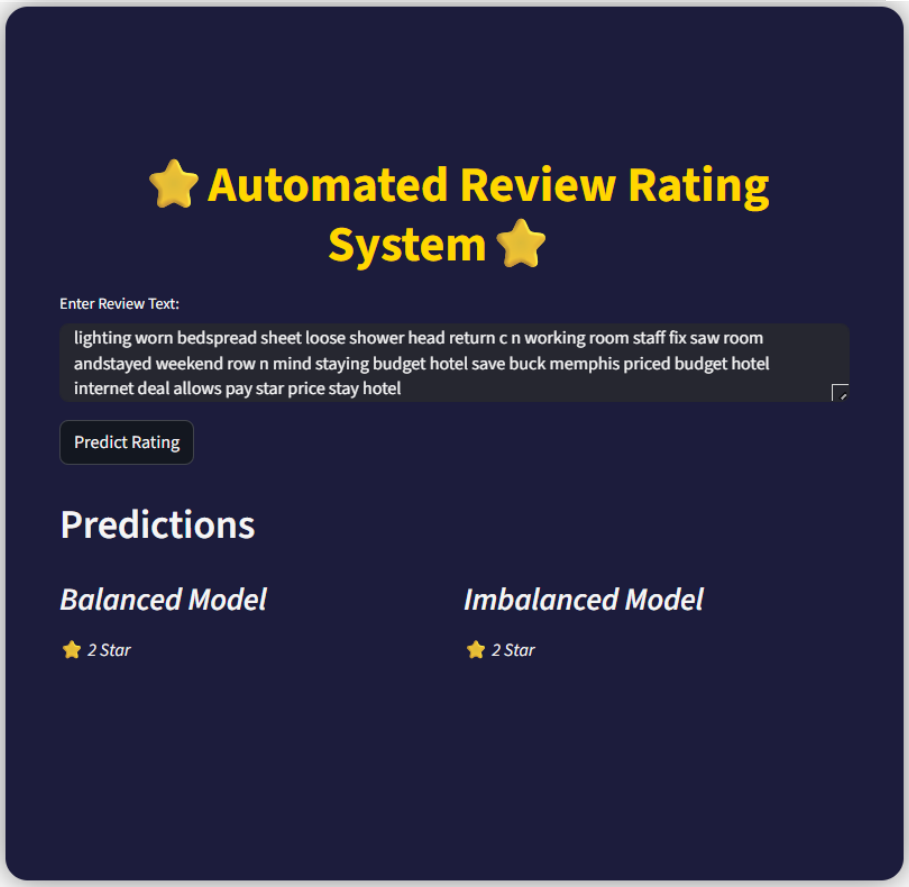


The system does not predict ratings for:

Empty reviews
Inputs containing only numbers or symbols

When such input is entered, it shows an error message instead of predicting.
This helps prevent invalid or meaningless predictions.


⭐ Automated Review Rating System ⭐
Enter Review Text:
model aircraft toilet shower room poor lightning ifficult read bed ask staff help safe deposit box bring torch air conditioning worked night daily complaint sewage smell bathroom place florence ok night euro night grateful wife share microscopic room
Predict Rating
Predictions
Balanced Model
⭐ 3 Star
Imbalanced Model
⭐ 3 Star


⭐ Automated Review Rating System ⭐
Enter Review Text:
lighting worn bedspread sheet loose shower head return c n working room staff fix saw room andstayed weekend row n mind staying budget hotel save buck memphis priced budget hotel internet deal allows pay star price stay hotel
Predict Rating
Predictions
Balanced Model
⭐ 2 Star
Imbalanced Model
⭐ 2 Star

This Streamlit app allows users to input a review and receive rating predictions from two different models for comparison. The app includes custom styling for a visually appealing interface. The prediction logic handles basic input validation and displays results neatly.

The picture shows predictions and confusion matrices from both models:

- Model A – Logistic Regression (Imbalanced)
- Model B – SVM (Balanced)

It was found that the SVM model performs better overall, as it predicts all rating classes (1–5 stars) more evenly. Even though Logistic Regression had slightly higher accuracy, it was more biased toward certain ratings, while SVM handled every class fairly.

SELECTING THE FINAL MODEL VERSION

After testing multiple models (logistic regression on imbalanced data and SVM on balanced data), the SVM model trained on balanced data was found to predict more accurately and reliably**.**

Therefore, this updated version of the app uses only the SVM balanced model for prediction.

Correspondingly, the code:

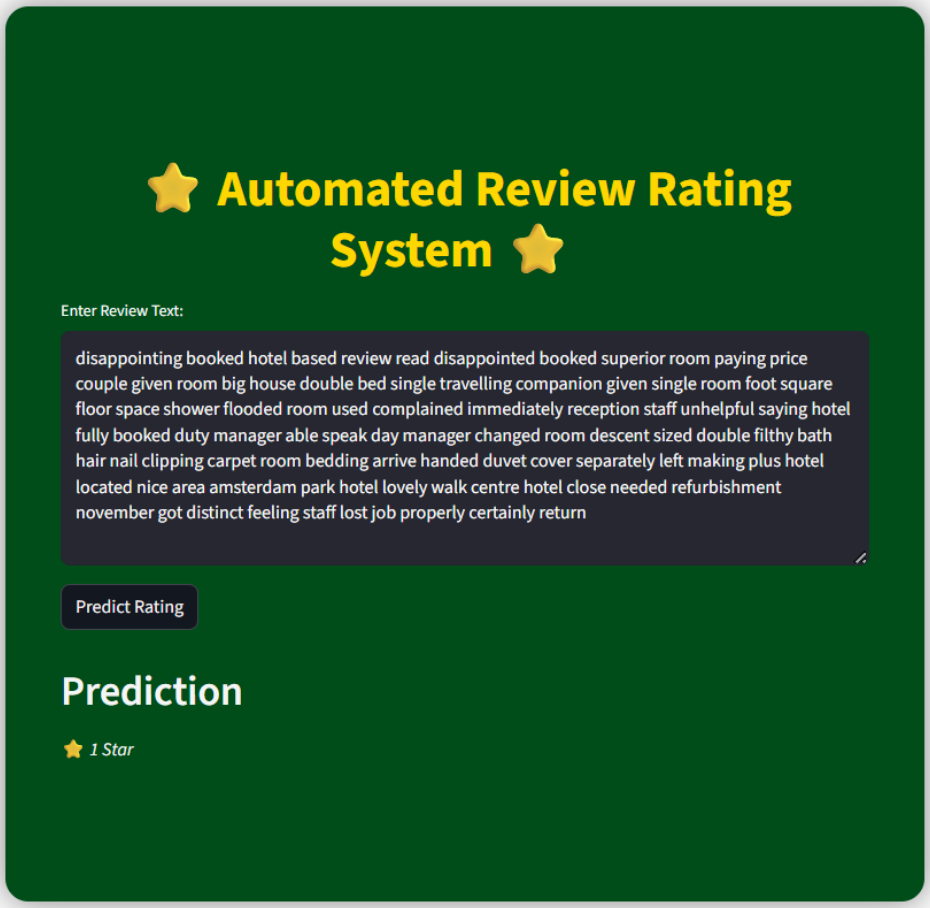
- Loads only the SVM model and its vectorizer.
- Removes other model loads and predictions to simplify the app.
- Displays only one predicted rating instead of comparing multiple models.

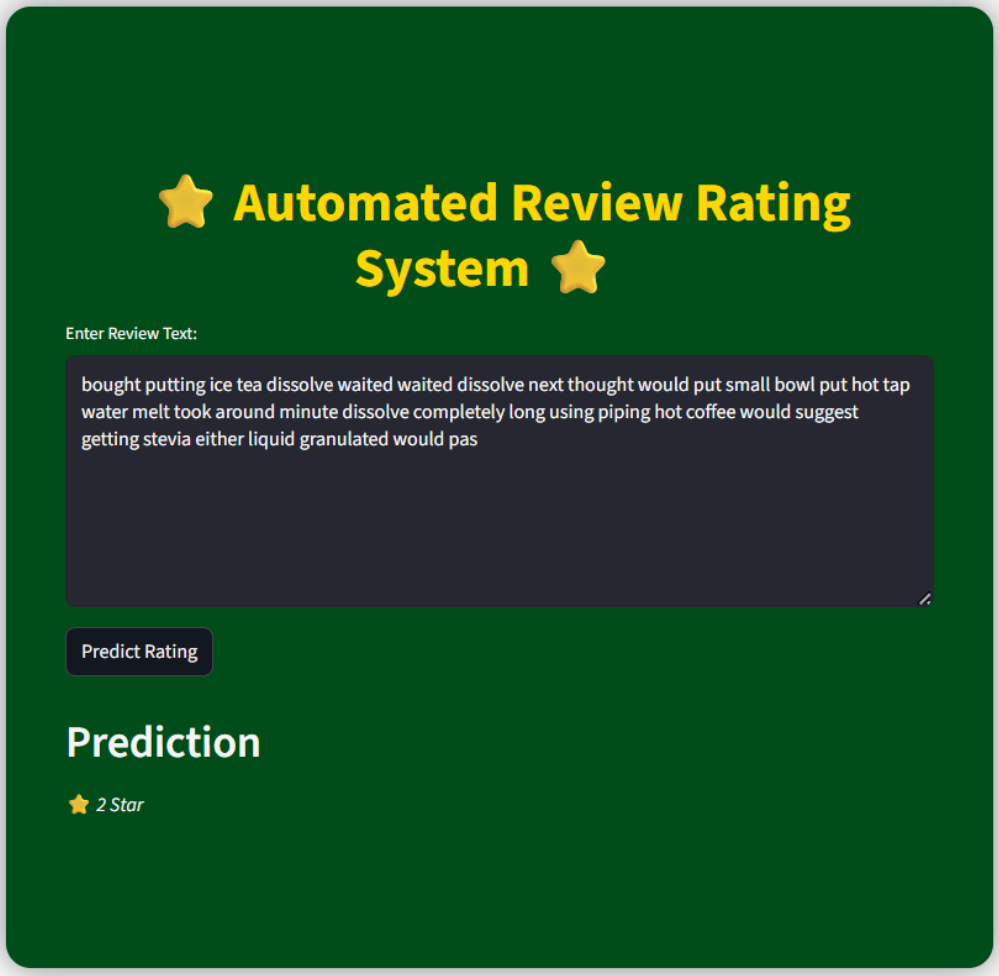
THE FINAL CODE OF UI

```
import streamlit as st
import joblib
model = joblib.load("models/fine_tuned_svm_balanced.pkl")
vectorizer = joblib.load("models/vectorizer_B.pkl")
st.set_page_config(page_title="Automated Review Rating System",
page_icon="⭐", layout="centered")

st.markdown("""
```

```
<style>
#MainMenu {visibility: hidden;}
footer {visibility: hidden;}
header {visibility: hidden;}
body{
background: linear-gradient(135deg, #1e3c72, #2a5298); /* blue background */
color: #f5f5f5;
font-family: 'Poppins', sans-serif;
}
.stApp{
background: #004d1a; /* dark green box */
color: #f5f5f5;
padding: 50px;
border-radius: 20px;
box-shadow: 0px 0px 30px rgba(0,0,0,0.5);
max-width: 800px;
margin: 50px auto;
}
h1{
color: gold !important;
text-align: center;
}
button[kind="primary"]{
background: gold !important;
color: black !important;
font-weight: bold !important;
border-radius: 8px !important;
}
</style>
""", unsafe_allow_html=True)

st.markdown("<h1>⭐ Automated Review Rating System ⭐</h1>",
unsafe_allow_html=True)
user_input = st.text_area("Enter Review Text:", height=150)
if st.button("Predict Rating"):
review = user_input.strip()
if review == "":
st.warning("Please enter a review to predict.")
elif not any(char.isalpha() for char in review):
st.error("Invalid review!")
else:
X_vec = vectorizer.transform([review])
pred = model.predict(X_vec)[0]
st.markdown("## Prediction")
st.markdown(f"⭐ *{pred} Star*")
```

OUTPUT:


⭐ Automated Review Rating System ⭐
Enter Review Text:
disappointing booked hotel based review read disappointed booked superior room paying price couple given room big house double bed single travelling companion given single room foot square floor space shower flooded room used complained immediately reception staff unhelpful saying hotel fully booked duty manager able speak day manager changed room descent sized double filthy bath hair nail clipping carpet room bedding arrive handed duvet cover separately left making plus hotel located nice area amsterdam park hotel lovely walk centre hotel close needed refurbishment november got distinct feeling staff lost job properly certainly return
Predict Rating
Prediction
⭐ 1 Star


⭐ Automated Review Rating System ⭐
Enter Review Text:
bought putting ice tea dissolve waited waited dissolve next thought would put small bowl put hot tap water melt took around minute dissolve completely long using piping hot coffee would suggest getting stevia either liquid granulated would pas
Predict Rating
Prediction
⭐ 2 Star

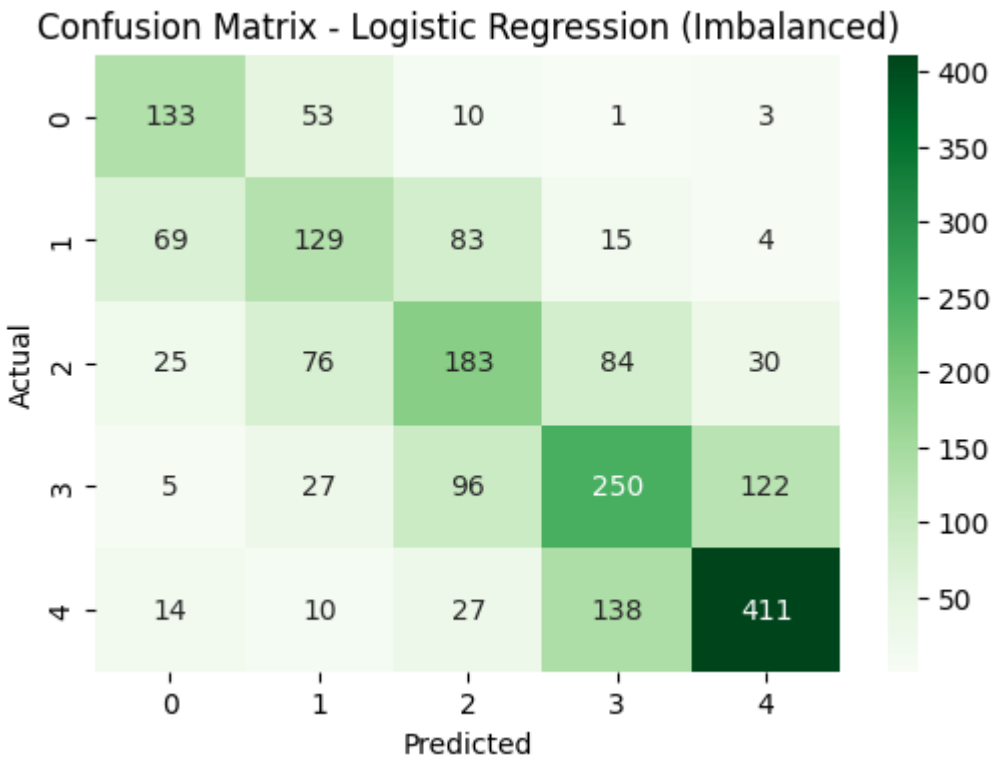
Results and evaluations:

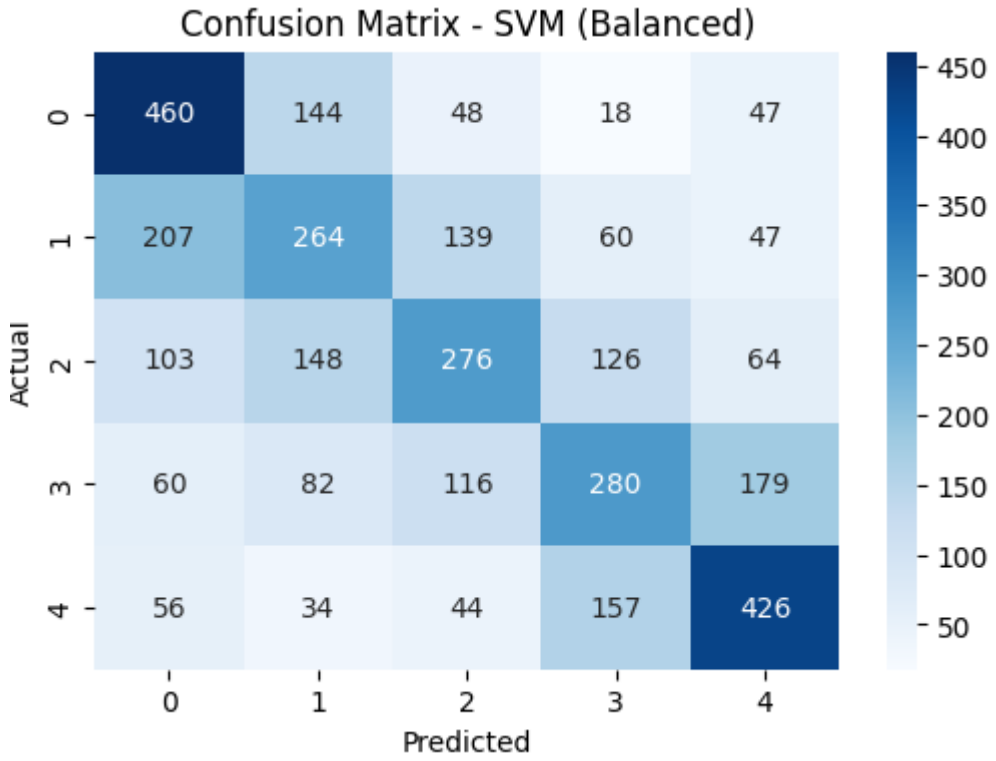
Although the Logistic Regression model achieved a slightly higher overall accuracy, it was observed to be biased toward majority classes (particularly 1 and 5 ratings).
In contrast, the SVM model demonstrated more consistent and fair predictions across all five rating categories (1–5).

Visualization

The confusion matrices for both models were analyzed to understand the class-wise performance.
From the visual comparison:

- Logistic Regression correctly identified most extreme sentiments (very negative and very positive reviews).
- SVM exhibited a more uniform distribution of predictions, minimizing bias and improving overall class balance.





These results indicate that SVM is more reliable when the goal is to predict every rating class fairly rather than focusing on accuracy alone

Final results:

Based on both quantitative and qualitative analysis, the SVM (Balanced) model was selected as the final model for deployment.

It provides a balanced, fair, and reliable performance suitable for real-world review rating prediction tasks.